[2pkt.] **Zadanie 3.**

Szablon rozwiązania: `zad3.py`

Dyrektor działu handlowego pewnej firmy odbywa podróż służbową z miasta A do miasta B. W pewnych punktach zaplanowanej trasy znajdują się stacje benzynowe. Niestety, ze względu na problemy z dostawami surowca, stacje limitują objętość paliwa, którą może zatankować pojedynczy klient. Co więcej, z powodu modyfikacji zmierzających do zwiększenia głośności i mocy silnika, samochód dyrektora spala aż 1 litr paliwa na 1 kilometr trasy. Dyrektor się spieszy - musi więc tak zaplanować podróż, by zatrzymać się na jak najmniejszej liczbie stacji. Jest to o tyle niełatwe, że każda stacja ma własny limit litrów paliwa, które można na niej zatankować. Dodatkową przeszkodą jest fakt, że w celu zmniejszenia masy pojazdu zmodyfikowano w nim zbiornik paliwa, który obecnie mieści jedynie `q` litrów benzyny.

Zaproponuj i zaimplementuj algorytm wskazujący na których stacjach dyrektor powinien tankować paliwo (tak, by tankować możliwie najmniejszą liczbę razy). Algorytm powinien być możliwe jak najszybszy i zużywać jak najmniej pamięci. Uzasadnij jego poprawność i oszacuj złożoność obliczeniową. Algorytm należy zaimplementować jako funkcję:

```
def iamlate( T, V, q, l ):
    ...
```

która przyjmuje:

1. Tablicę liczb naturalnych `T` z pozycjami stacji benzynowych, wyrażonymi jako kilometry od początku trasy. Pierwsza stacja znajduje się na początku trasy, t.j. `T[0]==0`. Kolejne stacje umieszczone są w `T` w kolejności odległości od początku trasy.

2. Tablicę dodatnich liczb naturalnych `V` zawierającą limity paliwa, które może zatankować pojedynczy klient. Tak więc `V[i]` to liczba litrów paliwa, którą można zatankować na stacji w pozycji `T[i]`. Na danej stacji można tankować tylko raz.

3. Dodatnią liczbę naturalną `q` będącą pojemnością baku samochodu (liczba litrów paliwa, które mieszczą się w baku). Zakładamy, że przed pierwszym tankowaniem w baku nie ma paliwa.

4. Dodatnią liczbę naturalną `l` będącą długością trasy w kilometrach.

Funkcja powinna zwrócić listę numerów stacji, na których należy zatankować paliwo (w kolejności tankowania). Jeśli warunki zadania uniemożliwiają dotarcie do celu, funkcja powinna zwrócić pustą listę. Stacje numerujemy od 0. Stacja na początku trasy stanowi część rozwiązania.

Przykład. Dla danych:

```
T = [0, 1, 2]
V = [2, 1, 5]
q = 2
l = 4
```

wynikiem jest np. lista: `[0, 2]`

[2pkt.] Zadanie 2.

Szablon rozwiązania: `zad2.py`

Dany jest spójny graf nieskierowany $G = (V, E)$. Proszę zaimplementować funkcję:

```
def breaking( G ):
  ...
```

która zwraca taki jego wierzchołek, że jego usunięcie (razem z incydentnymi krawędziami) powoduje że graf rozpadnie się na jak najwięcej nie połączonych ze sobą części. Funkcja przyjmuje graf reprezentowany przez kwadratową, symetryczną macierz sąsiedztwa i zwraca numer wierzchołka, z założeniem numeracji od zera. Jeśli usunięcie żadnego wierzchołka nie spowoduje tego że graf przestanie być spójnym, funkcja powinna zwracać `None`.

Funkcja powinna być możliwie jak najszybsza. Proszę oszacować złożoność czasową i pamięciową użytego algorytmu.

Przykład. Dla grafu, zadanego macierzą:

```
G = [[0,1,0],
     [1,0,1],
     [0,1,0]]
```

prawidłowym wynikiem jest 1.

[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`

Drzewo BST T reprezentowane jest przez obiekty klasy `Node`:

```
class Node:
  def __init__( self ):
    self.left    = None  # lewe podrzewo
    self.right   = None  # prawe poddrzewo
    self.parent  = None  # rodzic drzewa jeśli istnieje
    self.value   = None  # przechowywana wartość
```

Proszę zaimplementować funkcję:

```
ConvertTree(T)
```

która przekształca drzewo T na drzewo o minimalnej wysokości, w którym węzły spełniają warunek: największy element na danym poziomie jest mniejszy od najmniejszego elementu na kolejnym poziomie. Funkcja zwraca korzeń nowego drzewa. Poziomy numerujemy od korzenia do liści.

Funkcja powinna być możliwie jak najszybsza oraz—jako kryterium drugiego rzędu—używać jak najmniejszej ilości pamięci (poza pamięcią już wykorzystaną na reprezentację drzewa). Proszę oszacować złożoność czasową oraz pamięciową użytego algorytmu.

Przykład poprawnego przekształcenia.